UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería informática

TRABAJO FIN DE GRADO

MOOI, LA TERCERA EDAD EN CONTACTO

Leire Polo Martín

Tutor: Miren Idoia Alarcón Rodríguez

MAYO 2016

MOOI, LA TERCERA EDAD EN CONTACTO

AUTOR: LEIRE POLO MARTÍN

TUTOR: MIREN IDOIA ALARCÓN RODRÍGUEZ

Dpto. Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

MAYO 2016

Resumen

Hoy en día existe gran cantidad de personas de la tercera edad que pierden su independencia y la posibilidad de vivir solos por el miedo a que no recordar tomar la medicación pautada o a que pase algo y la persona no pueda pedir ayuda. Este problema se puede solucionar con cualquier método de tele-asistencia. Sin embargo, la realidad es que estos son caros y no están al alcance de todo el mundo.

Partiendo de esta realidad como motivación y, después de un estudio del estado del arte, el objetivo de este proyecto es el desarrollo de Mooi, un sistema de tele-asistencia para personas de avanzada edad **en el margen de la dependencia**, implementado como aplicación móvil híbrida y un servidor central.

El objetivo principal de Mooi es ofrecer un sistema de tele-asistencia gratuito y usable, por ello, Mooi está diseñado como un proyecto de software libre, disponible en la plataforma Github, de fácil empaquetado e instalación y con una documentación que facilita su utilización, es un proyecto fácilmente iterable para todo aquel que quiera añadir funcionalidades o construir sobre él, ayudando a mejorarlo.

Mooi ofrece, frente a otros sistemas, la facilidad de interacción a través de su interacción por voz, salvando la manipulación de una interfaz táctil, complicada en la mayoría de casos, para los usuarios de avanzada edad.

Para su desarrollo ha sido necesaria la implementación de un front-end o client altamente usable e intuitivo, desarrollado en las tecnologías HTML5, CSS3 y JavaScript, y los frameworks Ionic y Angularjs. Mientras que el desarrollo del back-end o servidor se ha realizado en Nodejs, desarrollando una API REST como método de comunicación entre front-end y back-end y una base de datos no relacional implementada sobre MongoDB. Este conjunto de tecnologías hace uso del famoso stack tecnológico MEAN, junto con las tecnologías más usadas en la programación de aplicaciones híbridas para móviles, así como los módulos complementarios que dotan a Mooi de la habilidad para comunicarse mediante la voz, e incrementar su entendimiento del usuario mediante el uso de redes neuronales gracias al módulo para Nodejs Natural Brain.

La satisfactoria validación y evaluación del sistema por parte de usuarios finales hace concluir que Mooi ha satisfecho con creces los objetivos planteados y la aplicación está operativa para su uso, si bien se han detectado mejoras para fases posteriores.

Palabras clave

Mooi, API REST, tele-asistencia, aplicación híbrida, aplicación móvil, aplicación web, Angularjs, Ionic, Nodejs, base de datos no relacional, frameworks de javascript.

Abstract

Nowadays there is a great amount of elderly people not able to live alone. They fear no to recall about their medication, as they fear to suffer some kind of incident or injury and not being able to ask anybody for help. This kind of inconvenients can be resolved with some kind of remote assistance, but assistance is expensive and it is not affordable by everyone.

Starting with this situation as motivation and after studying the state of the art, the idea of developing “Mooi” rises as this project objective. Mooi is a remote assistance system for dependant elders, in form of an hybrid app for mobile devices and central server.

Offering a free and simple remote assistance system is Mooi’s mains goal. That is why Mooi’s foundations are Open Source, its source code is available for free at Github. It is easy to distribute and install, it also haves documentation which helps it comprehension. It easy extensible and iterable, so anyone can help to its improvement.

Mooi offers as advantage amongst other systems, a spoken interface allowing to avoid in the vast majority of the cases the touch interface, which is hard to manipulate for the eldest users.

As said before, Mooi’s development is divided in two main areas. A highly usable and intuitive front-end or client written in HTML5, CSS3 and JavaScript with Ionic and Angular as frameworks. And a backend or server also written in JavaScript with nodejs. The server is built as a REST API as client-server communication layer and a persistence layer with the use of the non-relational database MongoDB. This is technologic stack is known as MEAN. Complimentary, Mooi uses modules allowing the recognition of the human voice and is able to learn by the use of neural networks implemented with “Natural Brain”.

The satisfactory evaluation and validation by the end users remarks the fact that Mooi satisfies its original objectives and its ready for its use, although there are some improvements already detected which will be developed in future releases.

Palabras clave

Mooi, API REST, tele-asistencia, hybrid application, mobile application, web application, Angularjs, Ionic, Nodejs, no relational data base, javascript. frameworks

Agradecimientos

Quiero agradecer a mi madre por decirme siempre que podía conseguir todo lo que me propusiese. A mi padre por enseñarme el amor por un trabajo bien hecho y por las ciencias. A mis hermanos por enseñarme a reírme de mi misma cuando la presión me superaba. A Álvaro por regalarme todo el conocimiento que ha hecho posible este proyecto. A mi tutora, por ofrecerme todo el apoyo y ánimo que necesitaba en esta etapa final de la carrera. A mis amigos y compañeros en esta etapa de mi vida, por ayudarme y hacer que mi vida sea mucho más divertida.

Índice de Contenidos

Figuras

Glosario

Javascript: Lenguaje de programación interpretado, usado para implementar la lógica tanto del lado del cliente como del lado del servidor.

Jquery: Biblioteca de Javascript.

HTML: *HyperText Markup Language*. Lenguaje de contenido para el desarrollo de páginas web.

JSON: *JavaScript Object Notation.* Estándar para intercambio de información.

API: *Application Programming Interface.* Interfaz de programación entre componentes software.

NoSQL: *No Structured Query Language*. Tipo de base de datos no orientada a relaciones, si no a documentos.

HTTP: *Hypertext Transfer Protocol*. Protocolo para acceso a la web.

Front-end: Parte del software que contiene la interfaz de usuario.

Back-end: Parte del software que procesa la entrada desde el front-end.

Framework: Estructura de soporte definida utilizada para organizar y desarrollar un proyecto software.

Angularjs: Framework JavaScript para el desarrollo de webs dinámicas.

Ionic: Framework de desarrollo de aplicaciones móviles híbridas.

1 Introducción

1.1 Marco del proyecto

El presente proyecto ha sido realizado íntegramente por la estudiante Leire Polo Martín, como propuesta personal de la propia estudiante. Dicho proyecto tiene como fin la implementación de una aplicación híbrida de tele-asistencia para personas de la tercera edad que conservan cierto grado de independencia. El desarrollo se ha realizado con herramientas avanzadas de programación web.

1.2 Motivación del proyecto

Hoy en día existe gran cantidad de personas de la tercera edad que pierden su independencia y deben dejar sus hogares para pasar al cuidado de especialistas en residencias de ancianos o bien al cuidado de familiares viviendo con ellos.

En la mayoría de casos, es el miedo a que pase algo y la persona no pueda pedir ayuda. Este problema se puede solucionar con cualquier método de tele-asistencia, el problema es que estos son realmente caros y no están al alcance de todo el mundo.

También se tiene miedo, normalmente justificado a que no se tome de manera adecuada la medicación, lo que deriva en una espiral descendente hacia la total dependencia, pues la falta de la medicación más simple si está pautada puede tener consecuencias fatales, por ejemplo el no tomar la medicación que controla la tensión puede provocar picos en la tensión lo que provoca micro infartos en el cerebro, lo que deriva en una demencia senil grave.

Es un hecho que la mayoría de los ancianos que dejan sus hogares por falta de independencia caen en depresión, ya que se enfrentan al hecho de verse incapaces de vivir por sus propios medios y además se ven forzados a dejar el lugar en el que han vivido.

Lo que pretende el desarrollo libre de MOOI, la propuesta de este trabajo, es obtener una aplicación de tele-asistencia y control gratuito que pueda funcionar con el 100% de la cuota de mercado de dispositivos Android, que salve la brecha digital mediante una interacción por voz y que pueda alargar el periodo de independencia y, por tanto, la calidad de vida de la personas de la tercera edad en la medida de lo posible, un tiempo realmente valioso al final de una vida.

1.3 Objetivo

El objetivo de este Proyecto de Fin de Grado es la definición, implementación y validación de un sistema híbrido cuyo propósito es la creación tanto del *back-end* como del *front-end* de una aplicación de tele-asistencia, principalmente dirigida a personas de la tercera edad que viven solas.

El resultado de este trabajo es una aplicación de asistencia con la que el usuario podrá interactuar mediante voz, de manera que no necesita un aprendizaje largo ni tedioso. Por tanto, al necesitar ayuda lo único que tendrá que hacer será, bien decirlo en voz alta y la aplicación se encargará de comunicar a su responsable por mensaje, o alerta en caso de tener la aplicación responsable descargada, o bien presionar un botón podrá llamar a emergencias o a un familiar.

La aplicación también servirá como recordador de medicamentos pudiendo interactuar con el usuario para que éste afirme que se lo ha tomado mediante un botón tras su alarma y esta información podrá verla la persona con la aplicación encargada.

Los aspectos más destacables de la aplicación en cuanto a producto final se refiere, son su sencilla interacción y la funcionalidad que ejecuta a un coste ínfimo respecto al coste de sistemas actuales.

Si se examina sus características técnicas, este proyecto representa un claro ejemplo del stack tecnológico MEAN (Mongodb, Expressjs, Angularjs y Nodejs) basado en programación en el lenguaje de programación Javascript y las bases de datos no relacionales, junto con una serie de tecnologías y técnicas punteras como puede ser la programación híbrida para dispositivos móviles y la inclusión de redes neuronales en servidores con el fin de aumentar la versatilidad de una funcionalidad automáticamente.

1.4 Estructura del documento

La memoria está formada por las siguientes secciones:

En la S**ección 2** se hace un análisis del estado del arte. En este estudio se investigan los sistemas similares al propuesto en este trabajo, es decir, sistemas de tele-asistencia tradicionales de empresas de asistencia o incluso aseguradoras, así como aplicaciones móviles en los markets de los diferentes sistemas

operativos, que puedan tener una funcionalidad similar a alguna de las diseñadas para esta aplicación.

En la **Sección 3** se detallan los objetivos y funcionalidades buscados en este trabajo de fin de grado. En primer lugar, se detalla el objetivo principal de forma global para situar al lector en un escenario y, en segundo lugar, se describen las funcionalidades y los objetivos específicos necesarios para que el sistema se desarrolle con éxito.

En la **Sección 4** se define el sistema a desarrollar donde se describe la metodología seguida y las herramientas utilizadas para el desarrollo del mismo.

En la **Sección 5** se realiza el análisis y diseño del sistema. En la primera parte, el análisis, se describen los requisitos funcionales y no funcionales. En la segunda parte, diseño, se describe la arquitectura y las estructuras de datos de la aplicación. Además, se explican las funcionalidades del sistema de forma detallada.

En la **Sección 6** se muestra la implementación de este trabajo. Se detalla el proceso de desarrollo de toda la aplicación junto con la base de datos y la interfaz gráfica.

En la **Sección 7** se describen las pruebas desarrolladas en la validación y verificación del sistema para comprobar el correcto funcionamiento del mismo.

En la **Sección 9,** se evalúa el resultado final para determinar los beneficios que se han obtenido del proyecto creado.

En la **Sección 10** se describe las conclusiones obtenidas al final del proyecto y posibles líneas de trabajo futuras.

Por último, en la **Sección 11** se enumeran las referencias utilizadas en el desarrollo de esta memoria.

2 Estado del arte

2.1 Introducción

Tradicionalmente la tele-asistencia se ha empleado mediante telefonía, aunque no móvil y siempre ha contado con dispositivos externos al teléfono convencional como método de aviso de un determinado problema. Sin embargo, en los últimos años se han hecho grandes avances en tecnología que han permitido pasar de grandes dispositivos anclados en las paredes de las casas con botones de emergencia a collares o pulseras para enviar alarmas en el momento necesario.

Uno de los problemas de estos sistemas es la falta de efectividad lejos del hogar, ya que la mayoría de dispositivos limitan su funcionamiento dentro de los límites de la vivienda. También deben ir unidos a una línea de teléfono fijo y no aportan ninguna información sobre la emergencia, más que el mero hecho de haberse apretado un botón.

Actualmente con la proliferación de las aplicaciones móviles, se han desarrollado sistemas que intentan aproximarse a las funcionalidades requeridas de la tele-asistencia, con un éxito variable, las características habituales de las que carecen la mayor parte de las aplicaciones móviles son la usabilidad y la accesibilidad, ya que no tienen en cuenta la “brecha digital”, y ligan su uso a una interfaz que, en gran número de ocasiones, a las personas de avanzada edad les es desconocida y difícil de aprender a usar.

Como solución a esta problemática, este Trabajo de Fin de Grado tiene como objetivo la creación de MOOI, una aplicación web y móvil híbrida, que tiene como fin servir de dispositivo de tele-asistencia al usuario de avanzada edad mediante una interacción sencilla e intuitiva por voz. Para crear el mejor servicio posible, se va a realizar un análisis de un conjunto de aplicaciones y dispositivos de tele-asistencia cuyas funcionalidades sean similares a las definidas para este proyecto. Se analizarán tanto dispositivos tradicionales como aplicaciones móviles haciendo un estudio sobre, no solo su funcionalidad sino también su accesibilidad ya sea por lugar de residencia o precio.

2.1 Sistemas existentes

A continuación, se analizan los sistemas de tele-asistencia existentes, tanto aplicaciones móviles, como sistemas tradicionales basados en telefonía fija:

Teleasistencia Vodafone

Este servicio consiste en una aplicación instalada en un dispositivo móvil Vodafone, que funciona las 24 horas de día y permite solicitar ayuda en situaciones de emergencia sanitaria y social, seleccionando dicha ayuda en la aplicación [1].

Teleasistencia Vodafone ha sido seleccionada para su estudio por ser actualmente uno de los pocos sistemas de tele-asistencia adaptados en forma de aplicación móvil del mercado.

Este sistema tiene un coste de 22 €/mes IVA incluido, sin incluir la tarifa de datos necesaria que supone como mínimo 10€/mes adicionales y debe ser contratado con el operador en cuestión, ni el coste del dispositivo que debe ser un Smartphone de la compañía Vodafone.

Teleasistencia Vodafone tiene una aplicación en la cual se puede pedir ayuda mediante un botón de emergencia que envía un mensaje a Cruz Roja con la localización GPS del dispositivo.

La aplicación de tele-asistencia no está disponible en las store de aplicaciones móviles, si no que debe ser instalada manualmente en una sede de Cruz Roja.

Para acceder a la instalación de la aplicación el usuario debe ceder sus datos personales a la base de datos de Cruz Roja y Vodafone.

En definitiva Teleasistencia Vodafone tiene como ventaja empaquetado del sistema en forma de aplicación móvil, pero en su detrimento se trata de un servicio caro que conlleva a tener un dispositivo y tarifa de datos de la compañía Vodafone, a ceder la información del usuario a dos entidades diferentes y a visitar 2 sedes diferentes (Vodafone y Cruz Roja) para la instalación del sistema en un dispositivo móvil.

Además, las funcionalidades de dicho sistema no van más allá de un mero botón avisador que no da información alguna sobre la situación de emergencia más allá de la localización GPS.

SAR QUAVITAE

Teleasistencia Móvil

Este servicio consiste en la tradicional tele-asistencia basada en un dispositivo en forma de collar o pulsera con un botón que el usuario debe apretar si tiene alguna emergencia [2].

Como en la mayoría de servicios con esta fórmula se trata de un servicio 24 horas 365 días al año que en su versión más asequible incluye un sistema receptor instalado en la vivienda del usuario y uno o varios dispositivos emisores que el usuario puede portar y en caso de emergencia, pulsar para enviar una señal al sistema receptor que enviará una señal vía línea de teléfono fija, de alarma a la central de la empresa contratada, la cual intentará contactar con el usuario y de no poder contactar o una vez corroborada dicha emergencia avisará a los servicios de emergencia y a los contactos de emergencia del usuario.

Este primer servicio de precio asequible se puede obtener por menos de 20€ mientras que una versión mejorada con un sistema de GPS gracias a un dispositivo móvil puede contratarse por 23,75 €/mes IVA incluido, este sistema agrega un sistema de cercado GPS que selecciona una zona de segura y al salir el usuario de dicha zona alerta a la central de la compañía como si de una alarma se tratase.

Este tipo de sistemas tienen asociado el coste adicional de mantener una línea de teléfono fijo y límite en cuanto a la distancia que puede alejarse el usuario de su residencia para que el sistema funcione de manera óptima, además de mantener el problema por antonomasia de los sistemas tradicionales, la cantidad de información que el usuario puede transmitir en un momento de emergencia mediante los métodos tradicionales es muy limitada.

La mayoría de sistemas de tele-asistencia una vez reciben un aviso de emergencia intentan ponerse en contacto con el usuario vía telefónica, pero en muchas de estos casos la propia situación puede impedir al usuario responder una llamada de teléfono.

Telealarma de la comunidad de Madrid

Este sistema, ofrecido gratuitamente por la Comunidad de Madrid es un sistema tradicional de tele-asistencia dentro de la vivienda del usuario con emisores de alarma en forma de pulseras o colgantes con un botón de alarma [3].

En este caso el servicio es más limitado ya que pese a la pulsera o colgante su funcionamiento queda restringido a los límites de la vivienda, siendo imposible su funcionamiento en el exterior.

Para optar a dicho servicio gratuito se deben cumplir una serie de requisitos previos que deben ser expuestos ante la comunidad de Madrid, para ello se debe presentar un procedimiento para el reconocimiento de la situación de dependencia, así como disponer del Programa Individual de Atención (PIA) que especifique que el usuario requiere dicho servicio.

En caso de no tener acceso al reconocimiento de la situación de dependencia por parte de la Comunidad de Madrid el usuario tendrá que exponer su caso ante el Consejo de Servicios Sociales de su ayuntamiento.

Este proceso burocrático puede extenderse durante meses y nunca se tendrá la certeza de la prestación de dicho servicio hasta su adjudicación, la cual está sujeta a un número de plazas limitadas.

Además el usuario nunca estará exento de la tenencia obligatoria de una línea de teléfono fija en su domicilio.

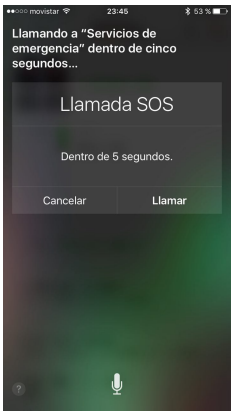
El hecho de tener en cuenta este sistema para el estudio previo realizado para el desarrollo de MOOI es que actualmente es el único sistema de tele-asistencia gratuito que se puede encontrar en España y pese a ser un sistema que lleva implantado varios años, aun así no atiende la demanda requerida.

SIRI/CORTANA

Tanto Siri (Apple), como Cortana (Microsoft) son aplicaciones que utilizan el procesamiento del lenguaje natural para realizar diversas acciones en los dispositivos móviles de los cuales son nativos[4] [5].

Sin ser propiamente sistemas de tele-asistencia han salvado la vida de muchas personas gracias a la ejecución de instrucciones por voz, como por ejemplo en 2015 cuando cumplió la orden de Sam Ray ,en Tennessee, de llamar a emergencias, cuando su coche le aplastó mientras lo arreglaba.

Estas aplicaciones que no han sido diseñadas para ser dispositivos de tele-asistencia tienen la cualidad más deseable de estos sistemas, la usabilidad, y es que dar instrucciones mediante la voz es la manera de interactuar más fácil, usable e intuitiva que se puede encontrar, y más cuando se tiene a un intérprete tan bien entrenado como son estas dos aplicaciones.



Se ha contado con estos dos sistemas para el estudio del estado del arte para el desarrollo de la aplicación MOOI ya que se tomará las interacción por audio como elemento diferenciador entre otros, frente a otros sistemas de funcionalidad similar.

2.2 Conclusiones

Tras el estudio del estado del arte, se han podido detectar necesidades no cubiertas por los sistemas actuales así como puntos de mejora de las actuales necesidades cubiertas.

En la mayoría de sistemas de tele-asistencia se aprecia una carencia de información considerable en una situación de alarma, ya que sin llegar a entablar comunicación telefónica es imposible transmitir detalles de la situación en cuestión.

Todos los sistemas de tele-asistencia son de pago con excepción del ofrecido por la comunidad de Madrid, y la cuantía supera en muchos casos el 5% de la pensión media Española, lo cual es un impedimento en aquellas personas que no poseen la solvencia económica necesaria para este gasto [6].

En el caso del sistema que ofrece la comunidad de Madrid, su concesión está sujeta a un cupo, y a unas condiciones, que no toda persona necesitada del sistema de tele-asistencia puede tener.

Todos los sistemas observados carecen de seguimiento del estado del usuario dependiente, siendo únicamente sistemas de alarma, y en ningún caso, monitorizando el estado del usuario fuera de estas situaciones.

3 Objetivos/Funcionalidades

En este punto se procederá a explicar el objetivo del sistema Mooi desarrollado en su conjunto, incluyendo los objetivos satisfechos por cada funcionalidad independiente.

3.1 Objetivos Genéricos

El objetivo de este proyecto es la definición, desarrollo y validación de un sistema de tele-asistencia, que se desarrollará en forma de aplicación móvil híbrida.

3.2 Objetivos específicos y Funcionalidades

En esta sección se procederá a describir los subsistemas del sistema Mooi. La división en subsistemas se ha llevado a cabo siguiendo un criterio funcional, así cada subsistema cubre un objetivo específico dentro del conjunto de funcionalidades implementadas en el desarrollo del sistema Mooi.

3.2.1 Alerta de emergencia

Mooi dispone de un sistema de reconocimiento de voz, con el cual el usuario dependiente puede, sin necesidad de interacción física, únicamente mediante la voz, dar la alarma de una situación de emergencia.

De esta forma Mooi recibirá los parámetros de la emergencia por voz al detectar ciertas palabras o conjuntos de estas, y en el caso de detectar un estado de emergencia se comunicará mediante voz con el usuario dependiente para obtener información sobre la situación y avisar al usuario responsable sobre esta.

3.2.2 Gestión de necesidades del usuario dependiente

Mooi dispone de un perfil de usuario dependiente y de otro perfil específico para el usuario responsable del usuario dependiente, en el cual dicho usuario puede modificar la programación de avisos para el usuario dependiente, que pueden ser tanto citas con el médico, como tomas de medicamentos.

Mooi dispone, como se especificará en el siguiente punto, de un sistema de traducción de texto a voz que se empleará en la notificación de dichas alarmas.

3.2.3 Recordatorio de medicamentos y alarmas

Mooi dispone de un sistema de traducción de texto a voz que permite reducir la interacción con el usuario dependiente a la interacción por voz y, gracias a la gestión de necesidades del perfil de usuario responsable, puede programar avisos de voz recordando las fechas de citas con el médico, así como la toma de medicamentos.

Pudiendo incluir una descripción de la alerta como por ejemplo “tómese el Aerius, la pastilla azul con la letra A”, se pueden reducir los errores en las tomas que suele ser un problema recurrente y peligroso en los pacientes de avanzada edad.

3.2.4 Seguimiento de histórico del usuario dependiente

Mooi, mediante el reconocimiento de voz y mediante una serie del algoritmos de comparaciones fonéticas es capaz de pasar a texto y almacenar un histórico de conversaciones y alertas, así como de respuestas a las diferentes preguntas programadas como la confirmación de las tomas de medicamentos, de manera que el usuario responsable pueda inspeccionarlo cuando desee.

3.2.5 Configuración de aplicación

Mooi dispone de una pantalla de configuración gracias a la cual se puede configurar el idioma de la aplicación, que será el idioma en el cual Mooi se comunique mediante voz y el idioma en el cual Mooi entienda las comunicaciones, respuestas o alertas que reciba.

En este panel también se tendrá acceso a los parámetros de configuración de usuario, nombre y contraseña.

3.3 Proceso funcional

El usuario responsable creará una cuenta introduciendo sus datos y los del usuario dependiente, desde entonces ambos podrán logarse en su dispositivo dentro de Mooi.

En usuario responsable se logará en la aplicación seleccionando en la pantalla de login si desea logarse como el tipo de usuario dependiente o como el tipo de usuario responsable e introduciendo su usuario y contraseña.

Una vez dentro del la aplicación tendrá acceso a un menú en el que podrá seleccionar entre diversas pantallas como la de información, configuración, alarmas e histórico de conversaciones.

El perfil de usuario dependiente al contrario solo posee una pantalla en la que contará con un botón de conversación y un botón de desconexión.

Mooi alertará de las alarmas programadas al usuario dependiente y transmitirá al usuario responsable todo lo que el usuario dependiente diga, y en caso de ser una alarma lo transmitirá de este modo.

4 Definición del Proyecto

4.1 Metodología

Para el desarrollo de Mooi se ha seguido un ciclo de vida en cascada iterativo como se puede ver en la figura 4.1. Este tipo de ciclo de vida se ha elegido debido al gran número de elementos a integrar, tanto en número de funcionalidades, como al gran número de librerías, módulos y frameworks utilizados [7].

Debido a la complejidad de integración de las diferentes partes y al avance natural de las tecnologías usadas en Mooi, ha sido necesario un ciclo de vida que permita las modificaciones controladas en fases avanzadas del proyecto, con el fin de obtener los mejores resultados en cuanto a funcionalidad e integrar adecuadamente las tecnologías elegidas tales como Ionic (se verá más adelante).

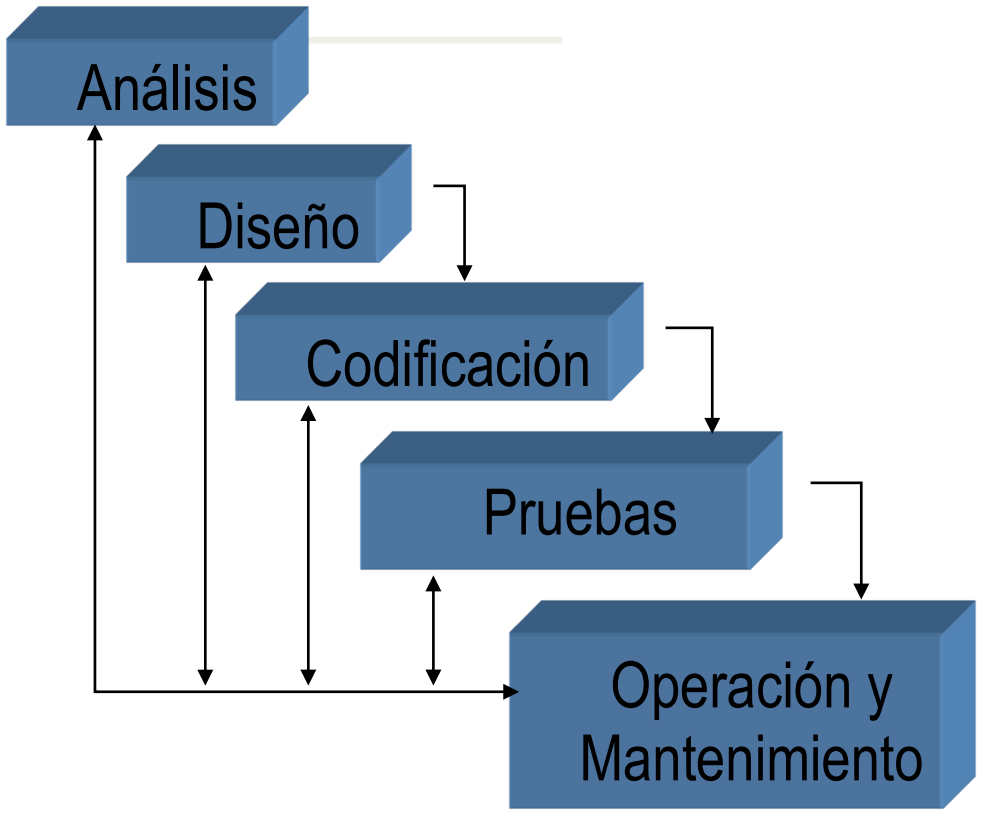


Figura 4.1: Ciclo de vida en cascada iterativo.

A continuación se resumen las tareas realizadas en cada una de las diferentes fases del desarrollo:

ANÁLISIS

1. Estudio del estado del arte.
2. Definición de objetivos y funcionalidades.
3. Especificación de requisitos funcionales y no funcionales.

DISEÑO

1. Diseño de la arquitectura del sistema.
2. Diseño detallado de los módulos del sistema.
3. Diseño de las interfaces de usuarios.
4. Diseño de los tipos de documentos de la base de datos no relacional.
5. Diseño de la API

CODIFICACIÓN

6. Implementación de la base de datos: MongoDB
7. Implementación del front-end: HTML + Ionic + Angular
8. Implementación del back-end: Nodejs

Pruebas y VALIDACIÓN

9. Realización del plan de pruebas (ya iniciado en la fase de diseño)
10. Diseño de los casos de prueba.
11. Desarrollo de pruebas: Mocha y Chai.
12. Evaluación del sistema con los resultados obtenidos a partir de un caso real.

4.2 Herramientas usadas

Para el desarrollo de Mooi se han usado las siguientes herramientas:

- IDE de desarrollo de aplicaciones móviles nativas en Android: Android Studio
- IDE de desarrollo de aplicaciones móviles nativas en iOS: XCode
- Herramienta para la creación de aplicaciones multiplataforma: Cordova
- Sistema de datos NoSQL: Mongodb

- Gestor de bases de dato no relacionales: Robomongo
- Lenguaje de programación de desarrollo de interfaces web: HTML5
- Lenguaje de hojas de estilos: CSS3
- Lenguaje de programación interpretado: JavaScript
- Biblioteca de JavaScript para manipulación del DOM: jQuery
- Entorno en tiempo de ejecución multiplataforma: Node.js
- Framework de diseño web: Angular.js
- Framework de desarrollo de aplicaciones multiplataforma: Ionic
- Notación de objetos en javascript: JSON
- Plugin de reconocimiento de voz: SpeechRecognizer
- Plugin modulador de voz: Text-to-Speech
- Módulo para Nodejs de relación del lenguaje natural: Natural Brain
- Módulo de documentación para Nodejs: APIDOC

En las próximas secciones se describen brevemente las herramientas mencionadas,

4.2.1 Android Studio

Android Studio es un IDE (entorno de desarrollo integrado) para el desarrollo de aplicaciones móviles [8].

En el desarrollo de Mooi ha sido necesario este IDE para facilitar la integración de la aplicación en dispositivos móviles con sistema operativo Android, ya que la depuración en de aplicaciones híbridas resulta compleja si no se cuenta con el IDE adecuado para el sistema operativo.

4.2.2 XCode

XCode es un IDE para el desarrollo de aplicaciones nativas de Apple, del mismo modo que en el caso de Android Studio ha sido necesaria su utilización para la depuración de la aplicación en su versión en iOS [9].

4.2.3 Cordova

Cordova es un marco de desarrollo de móvil de código abierto, el cual permite la utilización de tecnologías web para desarrollo de aplicaciones multiplataforma de manera que se pueda evitar el lenguaje nativo y en cierta parte la repetición del desarrollo para cada sistema operativo [10].

4.2.4 Base de datos

Para la creación de la base de datos se ha usado el sistema Mongodb, un sistema de datos NoSQL orientado a documentos, de manera que la información no se guarda en registros, si no en estructuras de datos similares a JSON llamado BSON con un esquema dinámico, lo que aporta mayor libertad a la hora de

insertar documentos y mayor facilidad de integración, tanto con otros sistemas, como con nuevas funcionalidades del sistema actual [11].

4.2.5 Desarrollo y manejo de la base de datos: Robomongo

Para facilitar la gestión y creación de la base de datos se ha utilizado Robomongo, un gestor de bases de datos específico para Mongodb que hace más intuitivo el manejo y más rápido la inserción de información y cambios derivados de los requerimientos del desarrollo [12].

4.2.6 HTML5

El código del front-end de la aplicación se ha implementado en lenguaje HTML5. Este lenguaje es la última versión del estándar HTML a cargo de la W3C y es la base para la creación de páginas web.

La aplicación se ha realizado incluyendo este lenguaje ya que actua como base para la realización de aplicaciones multiplataforma.

4.2.7 CSS3

Este lenguaje se usa conjuntamente con HTML para definir la presentación de del contenido especificado en HTML.

4.2.8 Javascript

Es un lenguaje de programación interpretado, basado en el estándar ECMAScript.

Pese a que su uso se limita en la mayoría de los casos al front-end de las aplicaciones web, en Mooi desempeña toda la lógica tanto en el front-end como en el back-end de la aplicación.

En el front-end realiza un papel clave para conseguir una aplicación en una sola página Single-Page Applications), siendo mucho más eficiente que sus homólogos de carácter más estático.

En el back-end implementa el servidor gracias a la plataforma Node, basada en el intérprete de JavaScript de Google Chrome, lo que le hace un sistema versátil que puede funcionar en los sistemas operativos más comunes, como son Windows, Linux o MacOS.

4.2.9 JQuery

JQuery es una biblioteca de JavaScript, creada para simplificar la interacción con los elementos contenidos en el HTML, de manera que facilite el manejo del árbol DOM y el manejo de eventos.

4.2.10 Nodejs

Es una librería y entorno de ejecución dirigida por eventos, que se ejecuta sobre el intérprete de Javascript de Google Chrome [13].

Su diseño enfocado a la maximización de rendimiento y eficiencia, además de su comunicación no bloqueante y capacidad de manejar eventos asíncronos lo convierten en la mejor opción para aplicaciones que manejan datos en tiempo real como es el caso de Mooi.

4.2.11 Angularjs

Es un framework MVC (Modelo, Vista, Controlador) de JavaScript para el desarrollo front-end de aplicaciones SPA(Single-Page Applications) [14].

El motivo de seleccionar este framework y no otro para el desarrollo de Mooi, es que al ser MVC, permite la separación de conceptos, dentro de los framework MVC concretamente Angularjs es la facilidad para extender el vocabulario HTML con directivas y atributos, manteniendo la semántica y sin necesidad de emplear librerías externas.

4.2.12 Ionic

Es un framework gratuito y libre para el desarrollo de aplicaciones híbridas multiplataforma que utiliza HTML5, CSS y Cordova como base [15].

Utiliza Angularjs para gestionar las aplicaciones, asegurando su escalabilidad y rapidez.

Una de sus ventajas particulares en la utilización de clases CSS3 y directivas de Angularjs que permiten el paso de un sistema operativo a otro mediante Cordova guardando una apariencia propia de aplicaciones nativas del propio sistema operativo.

4.2.13 JSON

JSON (JavaScript Object Notation) es un formato de datos, con una sintaxis dedicada que se usa para identificar y gestionar datos .

Es una alternativa al XML muy popular ya que puede ser leído por cualquier lenguaje de programación, y en caso de que se agregasen otros sistemas, tanto como proveedores de información como clientes del servidor de Mooi, no habría que alterar el formato de datos a transferir.

4.2.14 SpeechRecognizer

Es un módulo de Cordova que permite la captura de sonido e interpretación de diálogo [16].

Este módulo es compatible con sistemas operativos Android e iOS, lo que aporta versatilidad al código al no tener que iterar entre diferentes sistemas de reconocimiento de voz nativos.

4.2.15 Text-to-Speech

Es un módulo de Cordova que permite el modulado de voz a partir de texto [17].

Del mismo modo que el módulo de reconocimiento de voz es funcional en sistemas operativos Android e iOS.

4.2.16 Natural Brain

Modulo para Nodejs que, mediante redes neuronales ofrece una relación de oraciones en lenguaje naturales mediantes palabras que hacen de nexo, formando una red de relaciones entre oraciones [18].

4.2.17 APIDOC

Módulo de Nodejs para la documentación de APIs, lo que permite exponer mediante una página web, de manera visual y fácil de consultar la información necesaria para el uso correcto y de la API documentada [19].

5 Análisis y Diseño

5.1 Análisis

Tras un análisis de las necesidades de los usuarios finales, del estado del arte y del estado de integración de los usuarios con las nuevas tecnologías y sus formas de adaptación a éstas, se procede al análisis de los requisitos, tanto funcionales, como no funcionales, necesarios para el correcto desarrollo del sistema Mooi. A continuación se listan aquellos requisitos más relevantes para el sistema.

5.1.1 Requisitos Funcionales

RF.1 Mooi permitirá el registro de usuarios mediante una serie de datos personales básicos necesarios para la interacción futura y el acceso a la aplicación.

RF.2 Mooi permitirá logarse a los usuarios tanto responsables como dependientes desde una misma pantalla de login, pudiendo seleccionar en ella el modo en el que deben acceder.

RF.3 Mooi contará con dos tipos de interacciones dependiendo de si se ha logado como usuario responsable o como usuario dependiente.

RF. 4 Mooi permitirá al usuario dependiente la interacción por voz entablando conversación en caso de que el usuario diga una palabra desencadenante de interacción. En caso negativo, la traducción de la voz del usuario se guardará para su consulta posterior.

RF.5 Mooi permitirá al usuario responsable la modificación de las preferencias de idioma de comunicación e interpretación a través de un panel de control.

RF.6 Mooi permitirá al usuario responsable la creación de alarmas, tanto de toma de medicamentos como de citas del médico o recordatorios útiles para el usuario dependiente.

RF.7 Mooi permitirá al usuario responsable la eliminación de alarmas, tanto de toma de medicamentos como de citas del médico o recordatorios útiles para el usuario dependiente.

RF.8 Mooi permitirá al usuario responsable el acceso a un histórico de conversación del usuario dependiente en el que se guardarán todas las conversaciones o peticiones que el usuario dependiente haya hecho al sistema.

RF.9 Mooi enviará una notificación al usuario responsable en el caso de que una conversación o comportamiento del usuario dependiente desencadene dicha acción.

RF.10 Mooi comunicará las alarmas configuradas por el usuario responsable al usuario dependiente mediante un sonido y voz interpretando el texto prescrito por el usuario responsable.

5.1.2 Requisitos No funcionales

RNF.1 La aplicación realizará notificaciones al usuario responsable en menos de 30 segundos después de ejecutarse la acción desencadenante de la notificación.

RNF.2 La interfaz de usuario dependiente estará adaptada a las necesidades derivadas de la edad y condición del usuario final, de manera que el número de botones y elementos de interacción será reducido.

RNF.3 La interacción con del usuario dependiente con Mooi se podrá realizar mediante la voz.

RNF.4 La interfaz de usuario responsable expondrá la información relevante de manera clara y concisa limitando la interacción para acceder a la información más relevante de manera directa.

RNF.5 El idioma de reconocimiento de la aplicación será iterable entre castellano e inglés.

RFN.6 La aplicación se podrá usar en cualquier dispositivo móvil iOS y Android, quedando excluidos, por tanto, aquellos con sistema operativo Windows Phone o Blackberry.

RFN.7 El usuario deberá identificarse mediante usuario y contraseña.

RFN.8 En caso de no poder transmitir información al servidor o no recibir respuesta de este la aplicación informará al usuario mediante mensajes de error.

5.2 Diseño

Despues del estudio de los requisitos tanto funcionales como no funciones y el estudio de los sistemas y tecnologías actuales, se procede a la presentación de la arquitectura seleccionada para el desarrollo del sistema Mooi la cual se puede apreciar en la figura 5.1. La exposición de la arquitectura incluirá la descripcción de la estructura elegida para el desarrollo del front-end y back-end del sistema así como el esquema seleccionado para la base de datos.

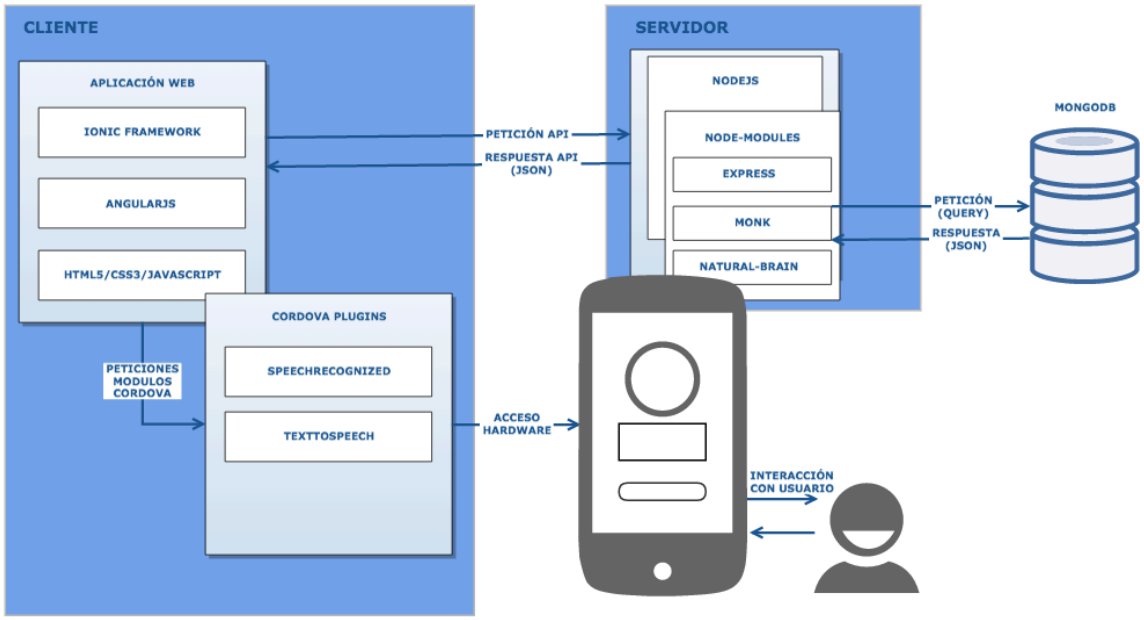


Figura 5.1: Arquitectura del sistema Mooi.

Los principales módulos que podemos observar son los siguientes:

- Base de datos documental, no relacional, en la cual se almacena la información necesaria para el funcionamiento del sistema, como por ejemplo la información de los usuarios, medicamentos e histórico de conversaciones de Mooi.

- Cliente de la aplicación para dispositivos móviles. Este módulo compone la parte residente en el dispositivo del usuario y es el que tiene contacto directo con él y se comunica con el servidor para la obtención de información de la base de datos ya que este módulo no mantiene conexión directa con ella. Del mismo modo, al tratarse de una aplicación

realizada con Angularjs, mantiene una estructura basada en MVC (modelo vista controlador), en la cual la lógica aplicada al cliente de la aplicación reside en los controladores mientras que las vistan mantienen la parte visual de la interfaz para el usuario.

- Interfaz del usuario, la cual permite al usuario dependiente mantener al usuario responsable informado de su situación y al usuario responsable la interacción con el sistema con el fin de monitorizar al usuario dependiente y programar los diferentes avisos que considere necesarios.

- API de tipo REST(Representational State Transfer), basado totalmente en el uso del protocolo http lo que permite la elaboración del servicios que pueden ser usados por cualquier aplicación que entienda http. La API tiene acceso directo a la base de datos y contiene una capa de lógica de usuario que decide la manera pertinente de actuación del servidor en cada situación.

5.2.1 Base de datos

La base de datos utilizada para el sistema Mooi, es una base de datos no relacional, basada en documentos de manera que la información se almacena en colecciones que son el homólogo a las tablas en una base de datos relacional.

Este tipo de base de datos se adapta perfectamente al ciclo de vida en cascada con realimentación utilizado, ya que mantiene laxitud a la hora de almacenar información, lo que hace más receptivo a este tipo de base de datos a cambios en etapas avanzadas del desarrollo.

Las colecciones de documentos son las siguientes:

- **Usercollection:** es la colección que almacena los documentos que contienen la información del usuario, tales como su nombre y contraseña, su id de referencia, los avisos programados, etc.

- **Pills:** es la colección que contiene la información de los medicamentos seleccionables en la aplicación, tales como su nombre, tipo de toma, descripción, prescripción, etc.

- **Conversation:** es la colección que almacena el histórico de conversaciones del usuario dependiente para su información al usuario responsable o bien para su posterior explotación.

- **IA:** es la colección que almacena la respuesta dada por la el servidor y la oración desencadenante comunicada por el usuario dependiente para su posterior estudio y explotación.

5.2.2 Interfaz de usuario

La interfaz del usuario se ha guiado por un diseño simple, limpio e intuitivo que minimice el esfuerzo del usuario en el proceso de interacción, ya que este punto es crítico al tratarse de usuario finales con dificultades en la interacción con tecnología en la mayoría de los casos.

5.2.3 API REST

La API se ha realizado en un servidor absolutamente independiente del cliente, lo que permite la integración de la misma API con otros clientes que deseen consumirla de manera a la hora de integrar nuevos sistemas, o simplemente en el caso de servir cliente y servidor en diferentes equipos este desarrollo lo permite sin problemas.

La API está compuesta por servicios necesarios para el funcionamiento del sistema Mooi, pero también contiene servicios de consulta a la base de datos de explotación de la información de investigación almacenada progresivamente, de manera que se pueda aprovechar para un estudio de funcionamiento del sistema o bien como parte de un estudio de comportamiento de los usuarios.

6 Implementación

6.1 Introducción

Se procede a la explicación tanto de la estructura y las tecnologías utilizadas, como de su integración dentro del desarrollo de los diferentes módulos que componen el sistema.

6.2 Estructura de la aplicación

El sistema mantiene una modularidad clara, componiéndose de un módulo en el cual se encontrará el front-end de la aplicación, o parte cliente, en la que reside el aspecto de la aplicación y una pequeña parte de lógica implicada en la interacción con el usuario.

El front-end de la aplicación ha sido desarrollado mediante la integración del framwork Ionic y la utilización de Angularjs haciendo uso de una estructura basada en MVC (Modelo vista controlador), siendo los controladores el almacén de la lógica residente en el cliente.

Del mismo modo, para poder desarrollar la aplicación híbrida apta para el uso en dispositivos móviles ha sido necesario el uso de Cordova, un marco de desarrollo de aplicaciones híbridas que proporciona una capa de acceso al hardware del dispositivo móvil contenedor, indispensable para la interacción de Mooi con el usuario dependiente mediante la voz.

El front-end no tiene acceso a la base de datos, de manera que para poder explotar la información residente en ella necesita interactuar con el módulo que contiene el back-end o servidor de la aplicación que sirve la API encargada de comunicarse con la base de datos y el cliente, exponiendo los servicios necesarios.

El back-end de la aplicación se ha desarrollado con Nodejs lo que hace posible el uso de JavaScript en este módulo de la aplicación, de manera que es posible la integración de módulos como el utilizado para desarrollar la inteligencia artificial encargada de decidir la lógica a emplear en las respuestas al cliente.

Dentro del back.end de la aplicación también se puede encontrar una capa de lógica de negocio encargada de seleccionar la línea de acción adecuada a las necesidades expuestas por el usuario que se alimentará de los módulos integrados gracias a Nodejs.

La comunicación con la base de datos se hará a través de Monk, otro módulo integrado en el back-end que facilitará la realización de peticiones a la base de datos para la explotación de su información.

6.3 Front-end

Como se puede observar en la figura 6.1, el front de la aplicación se ha realizado utilizando de manera conjunta el framework Ionic, que a su vez integra Angularjs y Cordova (lo que no exime de su utilización y conocimiento específico, pero facilita la integración de ambos), así como de las tecnologías HTML5, JavaScript y CSS3.

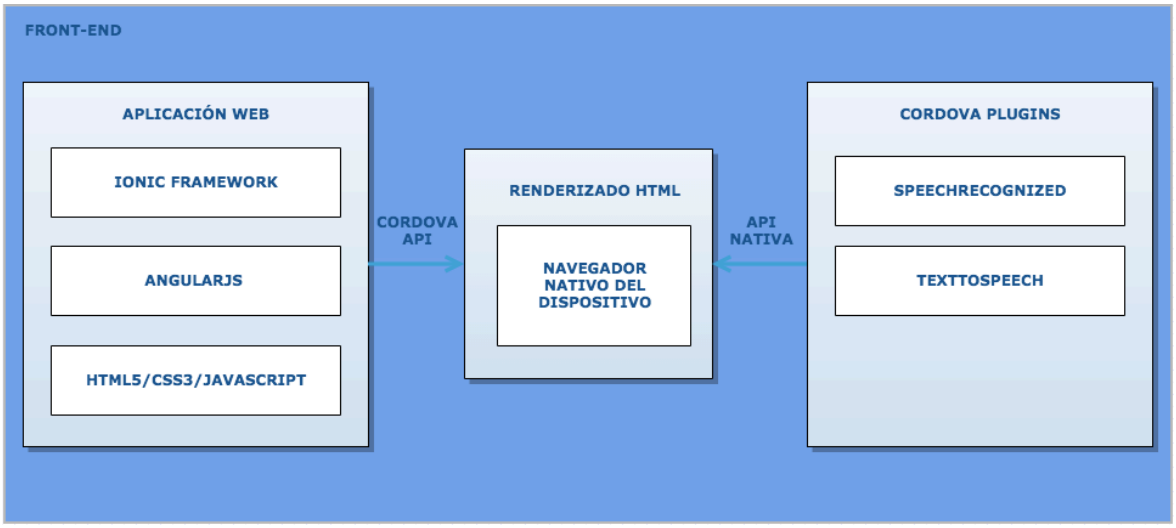


Figura 6.1: Arquitectura del front-end.

Mooi ha sido desarrollado como una aplicación en una sola página (Single-Page Applications), siendo siempre el mismo contenedor pero renderizando diferentes archivos html dentro de este.

La lógica es implementada dentro de los controladores, los cuales son ligados a determinados componentes pudiendo ligar, a su vez, determinadas acciones del

usuario como, por ejemplo, pulsar un botón o rellenar un formulario a las funciones desarrolladas dentro de estos controladores.

Aunque la mayor parte de la lógica se encuentra concentrada en los controladores, también parte de ella se puede encontrar en archivos para agrupar funcionalidades concretas como las llamadas a los servicios. Este es el caso de archivo ws.js que contiene las llamadas a los servicios de la API.

Como se puede observar en la figura 6.2, se ha estructurado el proyecto mediante un sistema de directorios basados en agrupaciones según tipo de elemento.

Todo el código de la aplicación esta incluido dentro del directorio www/, esta estructura es necesaria para el reconocimiento del código del proyecto en la integración mediante Cordova dentro de proyectos Android e iOS a posteriori.

Los archivos .css se agrupan en una carpeta con el nombre de su extensión para facilitar su localización a la hora de realizar cambios que afecten a la visualización de elementos.

Dentro del directorio img/ se almacenan las imágenes necesarias para la aplicación.

En el directorio templates/ se encontrarán los archivos html, que se integrarán como templates dando el aspecto visual de la aplicación, en cuanto a elementos visuales, ya que su visualización final dependerá de los archivos css.

Almacenando los controladores juntos, encargados de la lógica de funcionamiento, normalmente relacionada con la respuesta a acciones del usuario en su interacción con el usuario, se consigue una separación entre la parte visual, y la parte lógica del front-end, siendo una organización muchos más intuitiva a ojos de un desarrollador ajeno.

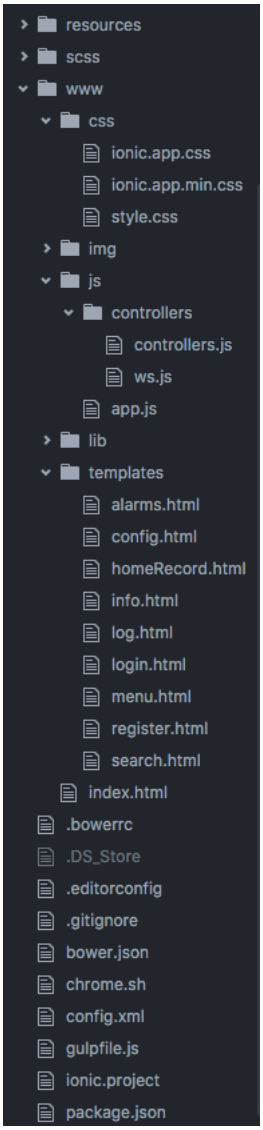


Figura 6.2: Estructura de directorios del back-end.

Siendo este un proyecto **open source** destinado a la colaboración futura mediante herramientas como Github, la estructuración clara del proyecto es crítica.

6.4 Back-end

Como se puede observar en la figura 6.3, el back-end de la aplicación consiste en una API REST desarrollada para comunicarse con el servidor, servir la información almacenada en la base de datos, así como para introducir en ella la información relevante.

La API ha sido desarrollada con la ayuda del módulo de Nodejs Express que facilita el desarrollo de manera considerable.Esta expone diferentes rutas tanto para obtención, como para inserción de información en la base de datos.

El punto más importante del back-end de Mooi es la capa de la lógica de negocio, la cual mediante el módulo NATURAL-BRAIN es capaz de entrenar a Mooi para aprender y aumentar su rango de entendimiento del lenguaje humano.

Mooi utiliza un sistema basado en relación de palabras del interior de las oraciones, para ser capaz de dar la respuesta correcta a una oración que no le fue programada manualmente, si no que aprendió mediante la relación de la nueva y desconocida pregunta con una pregunta anterior para la cual si tenía solución programada.

Esto convierte a Mooi en un sistema que autoescala el entendimiento de los usuarios, alimentándose de las respuestas de todos ellos.

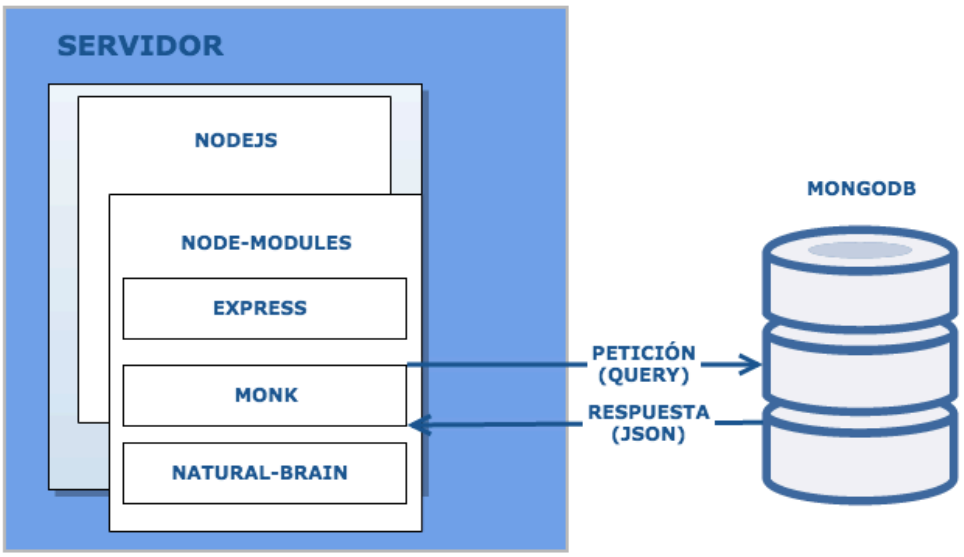


Figura 6.3: Arquitectura del back-end.

La estructura del “proyecto” back-end de Mooi, como se puede observar en la figura 6.4 es la siguiente:

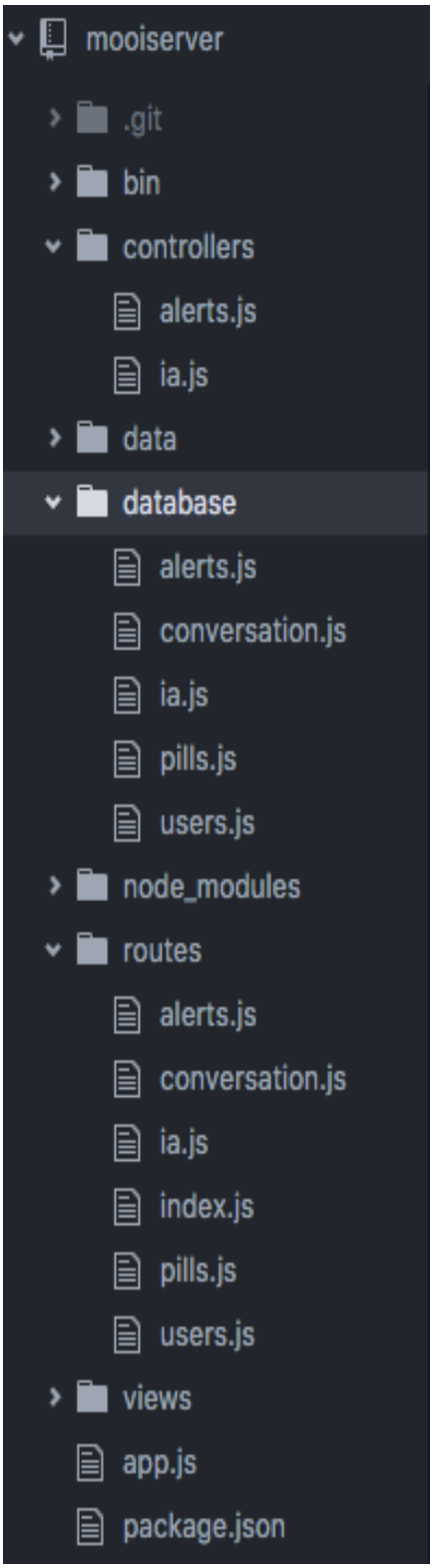


Figura 6.4: Estructura de directorios del back-end.

En primera instancia, toda petición a los servicios expuestos por la API pasarán por el archivo app.js.

Siendo la puerta de entrada a la lógica del servidor es el motivo por el cual se encuentra separado del resto de archivos y situado en la raíz.

App.js redirecciona según su enrutado a la ruta que corresponda dentro del directorio routes/ , donde se encuentra la distribución según su ruta.

Desde el archivo de ruta seleccionado se dirigirá el flujo hacia el controlador indicado, situado dentro del directorio controllers/ que es donde reside la lógica de negocio de la aplicación.

En caso de ser necesaria la interacción con la base de datos se debe acceder a las funciones implementadas dentro del directorio database/ donde se realizan los accesos a la base de datos mediantes el módulo Monk.

La estructura de esta parte del proyecto está basada en el flujo que deben recorrer las peticiones desde su entrada hasta su respuesta.

6.5 Base de datos

La base de datos se ha realizado en Mongodb , siguiendo un esquema no relacional que cuya estructura se puede observar en la figura 6.5.

El acceso desde el servidor a esta base de datos se realiza mediante el módulo Monk de Nodejs, para facilitar la manipulación y acceso a los datos.

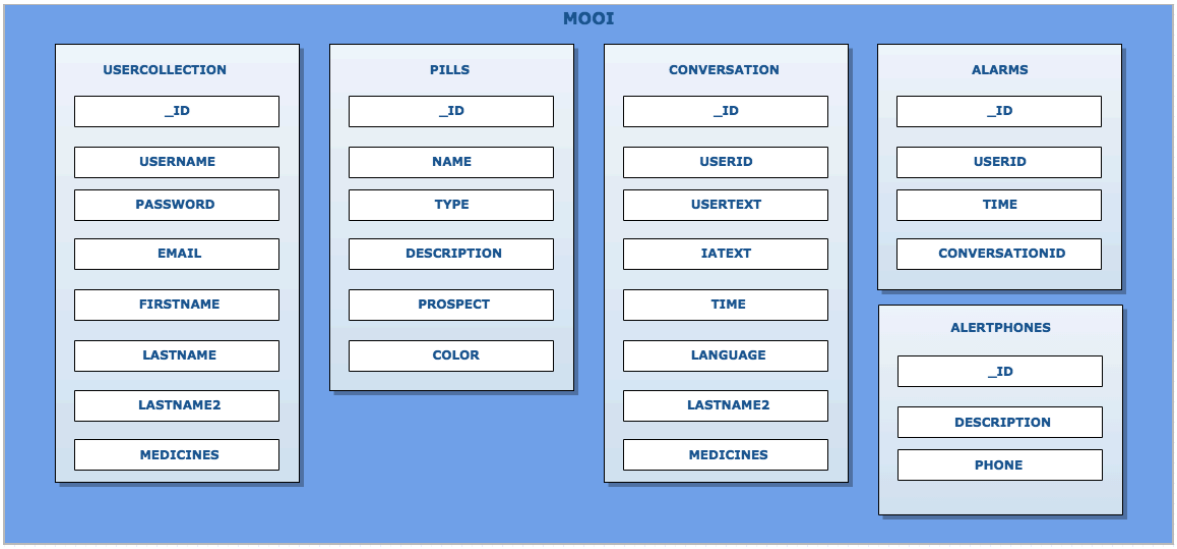


Figura 6.5: Estructura de la base de datos no relacional.

6.6 Interfaz de usuario

La interfaz del usuario ha sido desarrolladas con las tecnologías HTML5, CSS3 y Javascript, con los framework Ionic y Angularjs, empaquetado como aplicación híbrida para su utilización en dispositivos móviles mediante Cordova.

La interfaz de la aplicación es sencilla e intuitiva y se adecúa a las necesidades de los usuarios finales, haciéndolo usable.

6.6.1 Registro de usuario

En esta pantalla, como se ve en la figura 6.6 el usuario responsable podrá registrar tanto sus datos como los del usuario dependiente, creando una cuenta con la que poder logarse posteriormente en la aplicación para su uso.

Para el registro es necesario la aceptación de la política de uso de la aplicación en la que se indican la posibilidad del uso de los datos recolectados del uso de la aplicación.

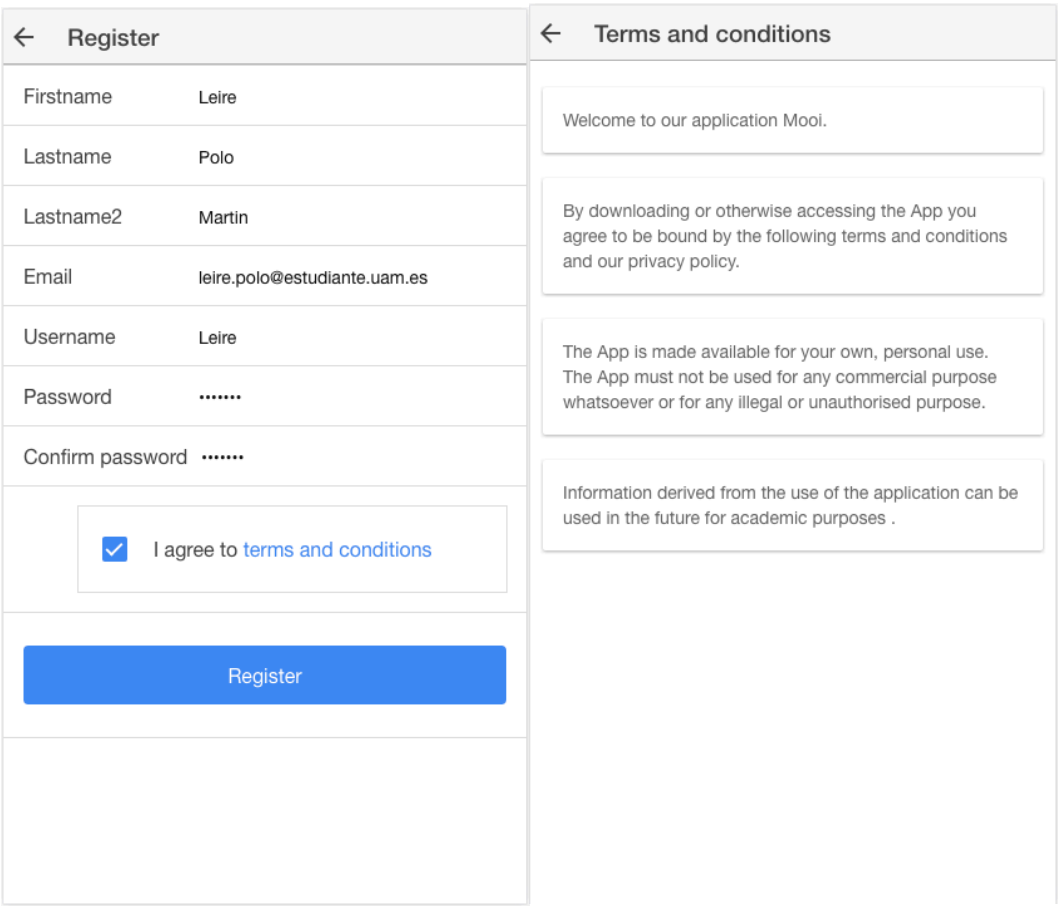


Figura 6.6: Pantalla de registro.

Figura 6.7: Pantalla de términos y condiciones de uso.

6.6.2 Condiciones de uso

En esta pantalla, como se ve en la figura 6.7 se pueden consultar las condiciones de uso que el usuario debe aceptar para completar su registro en la aplicación, su aceptación es crucial ya que la información recolectada durante el uso del sistema puede tener usos académicos a posteriori.

6.6.3 Login de usuario

En esta pantalla, como se puede ver en la figura 6.8, el usuario podrá logarse indicando con que tipo de perfil quiere acceder a la aplicación.

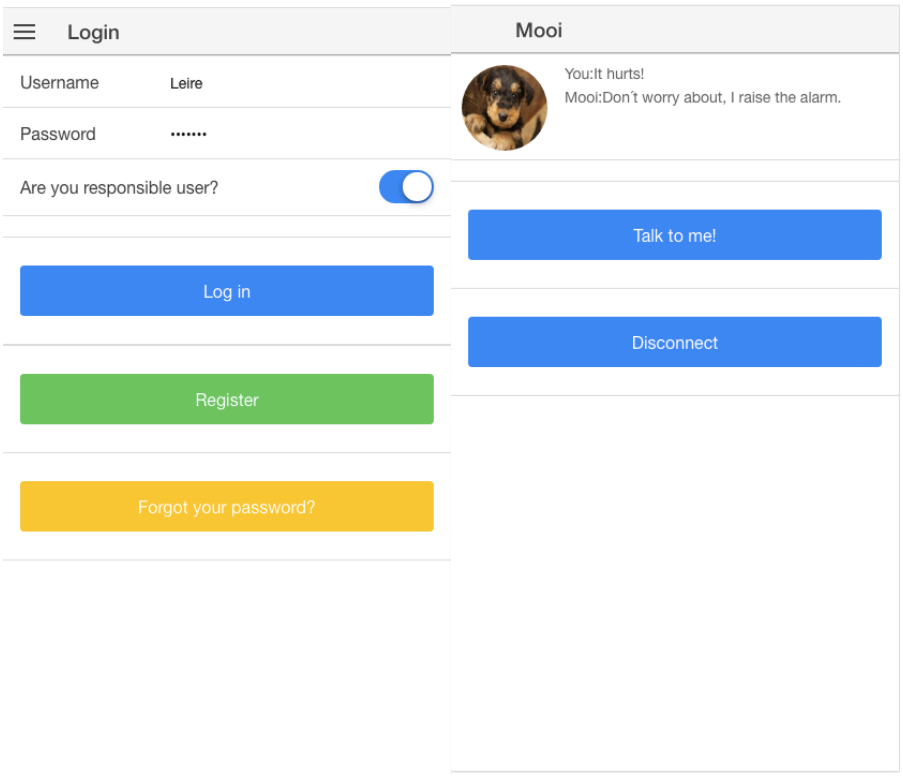


Figura 6.8: Pantalla de login.

Figura 6.9: Pantalla de usuario dependiente

6.6.4 Perfil de usuario dependiente

En esta pantalla, como se puede ver en la figura 6.9, el usuario dependiente podrá interactuar con Mooi y desconectar en caso de ser necesario.

6.6.5 Información

Desde el perfil de usuario responsable, el usuario puede acceder a esta pantalla, que como se puede observar en la figura 6.10, muestra la información que el usuario dependiente ha transmitido así como el listado de las alarmas que se hayan ido sucediendo.

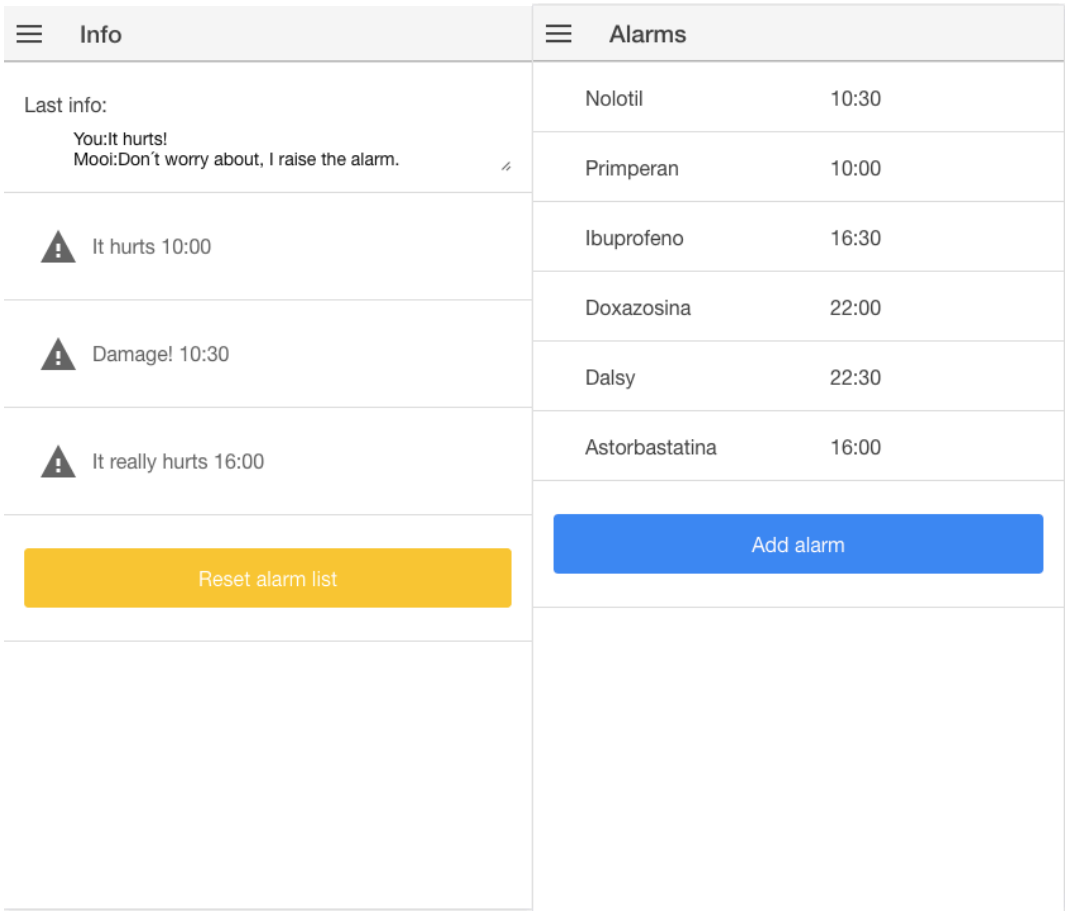


Figura 6.10:Pantalla de información

Figura 6.11: Pantalla de listado de alarmas

6.6.6 Listado de alarmas

Desde el perfil de usuario responsable, el usuario puede acceder a esta pantalla, que como se puede observar en la figura 6.11, permite visualizar y seleccionar las alarmas.

Seleccionando cualquiera de las alarmas se redirigirá a su pantalla de configuración.

6.6.7 Configuración de alarma

Como se puede observar en la figura 6.12, en esta pantalla se puede configurar las características de las alarmas, y asignar una descripción para más información.

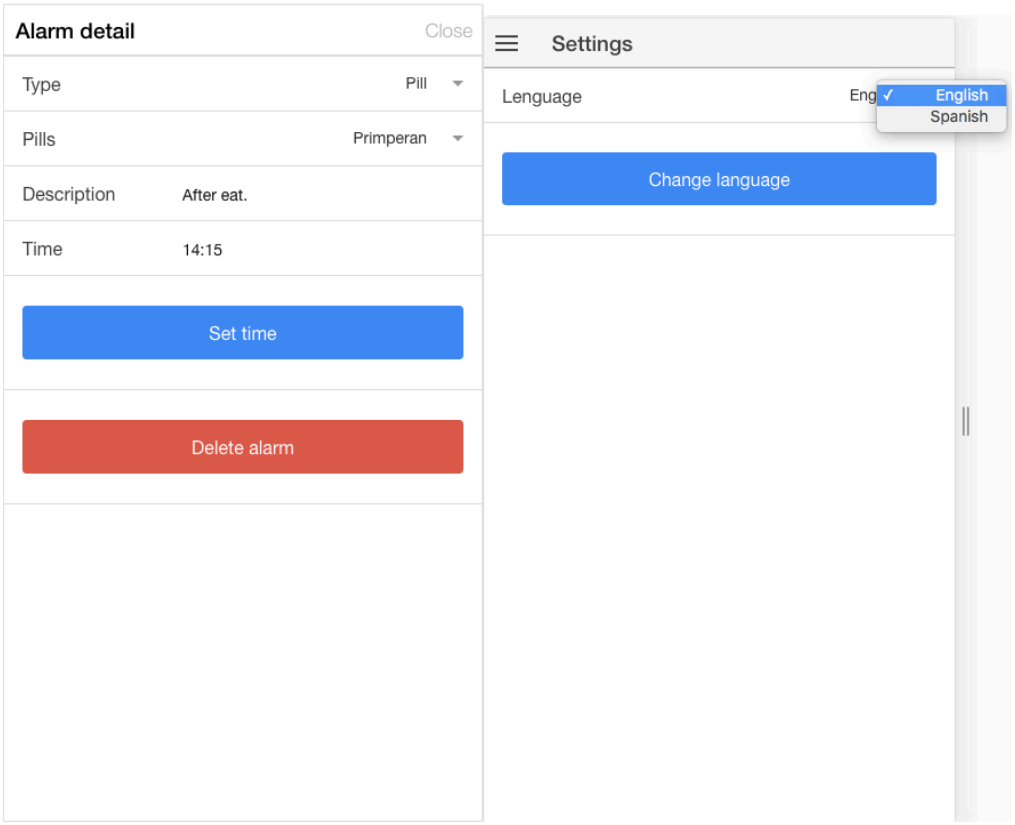


Figura 6.12: Pantalla de configuración de alarmas.

Figura 6.13: Pantalla de configuración.

6.6.8 Ajustes

En esta pantalla el usuario responsable puede, como se puede ver en la figura 6.13, cambiar el idioma de reconocimiento de voz de la aplicación.

6.6.9 Historial

En está pantalla, como se puede observar en la figura 6.14, el usuario responsable puede monitorizar la interacción del usuario dependiente con la aplicación, observando parte del histórico de conversaciones del usuario con Mooi.

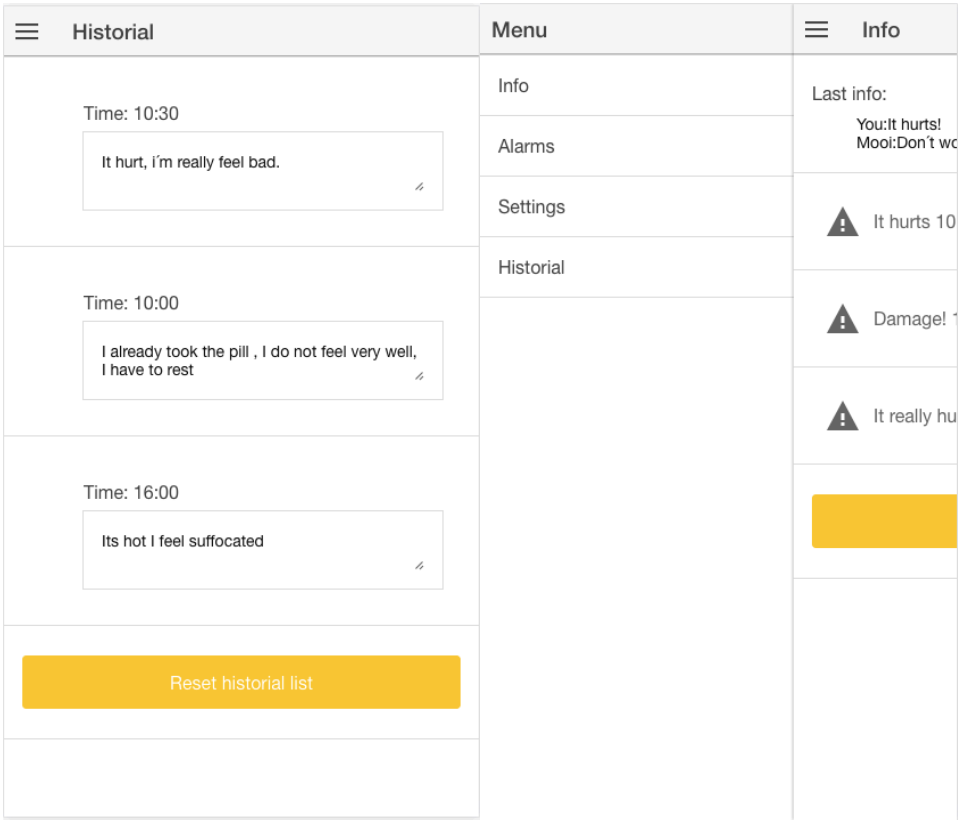


Figura 6.14:Pantalla de Historial

Figura 6.15: Menú desplegable

6.6.10 Menú

En las pantallas del perfil de usuario responsable te tiene acceso a un menú desplegable como método de navegación entre las posibles funcionalidades, como se puede observar en la figura 6.15.

7 Validación y Verificación

7.1 Verificación

Se ha realizado una verificación del código y validación de la funcionalidad con el objetivo de asegurar que los diferentes módulos que componen el sistema Mooi, así como las funciones individuales, funcionan correctamente, asegurando la integridad del sistema, y minimizando los riesgos en un sistema destinado a un fin para el cual este punto es crítico.

7.1.1 Estrategia de pruebas

La funcionalidad de la aplicación debe probarse en su totalidad, y siendo una aplicación en la que sus diferentes módulos tienen especificaciones muy distintas, cada uno se debe probar de manera diferente. La estrategia seguida ha sido una estrategia de dentro hacia fuera, comenzando con las pruebas unitarias, donde se han realizado pruebas a los distintos módulos y continuando con la integración de dichos módulos de manera ordenada hasta llegar a las pruebas del sistema completo.

En esta sección se describen los distintos tipos de pruebas, mientras que en la siguiente se especifican los resultados de las mismas.

Antes de describir los distintos niveles de pruebas, se hace hincapié en la inspección de código realizada, tarea que se ha considerado imprescindible en esta fase de pruebas.

Inspección de código

La inspección de código consiste, en la lectura del código por parte del desarrollador una vez terminada la implementación de la funcionalidad con ánimo de encontrar errores de lógica o sintaxis de manera visual, no por detectar un mal funcionamiento.

En este proceso se ha puesto especial interés en la revisión de la lógica de negocio del servidor, ya que es el punto más comprometido de la aplicación y en el que más lógica hay, pudiendo ser un punto de error que no produzca un error velado en la aplicación pero si un mal funcionamiento, siendo este caso más difícil de detectar.

Aunque es una prueba rudimentaria y poco segura, se ha realizado con la intención de ser exhaustivo, dado lo crítico en cuanto a errores del sistema.

Pruebas unitarias

Las pruebas unitarias se han realizado en cada módulo por separado, probando en cada caso las funcionalidades pertinentes.

Estas pruebas son especialmente útiles en el caso de la API ya que con el uso de la herramienta Postman se pueden probar de una manera “manual”, pero es fácil probarlas y comprobar su correcto funcionamiento.

Aunque se han realizado las pruebas usando la herramienta Postman, también se han realizado pruebas automáticas mediantes los módulos de pruebas en JavaScript Mocha y Chai, con lo que se implementan baterías de pruebas automáticas mediante la inserción de valores y especificación de funcionalidad correcta.

Al realizar el desarrollo se ha seguido la metodología TDD **Test-driven development**, un tipo de desarrollo guiado por test, de manera que los test unitarios automáticos se implementaron antes que la propia funcionalidad, aunque aumentándolos de manera incremental, lo que fue necesario debido a la realimentación del ciclo de vida seleccionado.

Estas baterías de pruebas son cruciales a la hora de realizar un proyecto con ciclo de vida es cascada, ya que la nuevas iteraciones sobre funcionalidades anteriores a menudo producen errores en funcionalidades anteriormente probadas.

Este proceso de automatización ahorra tiempo y asegura la detección de errores sobre código ya probado que produzcan nuevas iteraciones.

Pruebas integración

Las pruebas de integración están destinadas a encontrar errores de funcionalidad sobre funcionalidades que integran más de un módulo, de manera que al juntar dos o más módulos se deben realizar estas pruebas.

Del mismo modo que se han probado con anterioridad mediante pruebas unitarias los servicios de la API, una vez integrada la API con la base de datos se procederá a verificar si los datos devueltos por la API coinciden con los

esperados, estas pruebas también se han realizado de manera automática con Mocha y Chai.

Al integrar la el back-end con el front-end de la aplicación se han realizado pruebas del mismo modo, pero está vez revisando si la visualización de la información obtenida de la API es la correcta en el front-end.

El proceso de integración que se ha seguido en el desarrollo del sistema Mooi, tras el desarrollo independiente de front-end, API de comunicación, lógica de negocio del servidor y base de datos, ha sido el siguiente:

1. Integración de la API con la base de datos, comprobando el correcto funcionamiento de las llamadas que integraban consultas o modificaciones a la base de datos.
2. Integración de la API y base de datos con la lógica de negocio, comprobando la correcta inyección de dependientas de los dos sistemas conjuntamente.

Integración de front-end y back-end comprobando el funcionamiento conjunto de la aplicación.

Pruebas de interfaz

Las pruebas de interfaz se realizan con el objetivo de detectar la incorrecta visualización o incorrecto funcionamiento de algún elemento de la interfaz gráfica.

Este proceso pasa por minuciosa visualización de todas las pantallas y comprobación de los elementos interactivos, y en el caso de ser una aplicación híbrida este proceso debe realizarse en diferentes dispositivos con diferentes resoluciones, para asegurar la correcta visualización en el mayor abanico posible de dispositivos.

Pruebas de validación

Dada la importancia de este tipo de pruebas se explican en detenimiento en el siguiente capítulo.

Ya que todo el mundo no utiliza la aplicación de la misma manera, y nunca debe ser el usuario tester el mismo desarrollador se ha pedido la colaboración de usuarios más cercanos al perfil de usuario final.

La última prueba que se ha realizado sobre la aplicación ya integrada ha sido realizada por dos usuarios de 84 y 83 años de edad que han probado la aplicación

poniendo a prueba interacciones más naturales para el usuario destinatario del sistema.

7.1.2 Desarrollo de pruebas

A continuación se detallan los resultados obtenidos en las pruebas y errores detectados:

Inspección de código

Tras una detenida revisión del código, consistiendo ésta en la lectura detenida de la implementación se han detectado los siguientes errores, que posteriormente fueron solucionados:

- Errores de implementación con cierta repetición en las llamadas a funciones de consulta en la base de datos dentro del back-end de la aplicación, las llamadas ejecutan promesas y faltaba la devolución de la promesa.
- Errores de sintaxis en la función HomeRecordCtrl llamada al módulo Text to Speech con errores al invocar la función speack en lugar de speak.
- Error en la query de consulta en la base de datos al invocar la colección usercolllection en lugar de usercollectión.

Todos estos errores se han solucionado correctamente.

Pruebas unitarias

Al seguir la metodología TDD los test unitarios se realizaron antes de la implementación y se pasaron paulatinamente según avanzó el desarrollo eliminando la existencia de errores que detectases estos test al final del desarrollo.

Pruebas de integración

En las pruebas de integración se detectaron los siguientes errores, que se resolvieron en esta misma fase:

- Los errores de referencias en la API fueron los más comunes ya que al integrarse junto a la base de datos los nombres de referencia en muchos casos estaban cambiados.
- Se encontraron numerosos errores por falta de control de errores a la hora de establecer conexión con la base de datos, al no estar

controlados, al darse un error por fallo de la base de datos provocaba una caída descontrolada del servidor.

- Ausencia de control de errores por timeout en las llamadas a servicios de la API en el front-end.

Pruebas de interfaz

Tras revisar las interfaces en 3 dispositivos móviles de diferentes resoluciones y tamaños, se detectaron los siguientes errores:

- En tamaños pequeños de pantalla se detectó una aglomeración de los elementos importantes lo que llevo a la redimensionalización de elementos de la interface de usuario.

- Debido al exceso de sensibilidad en la pantalla táctil de los dispositivos de gama más alta se detectó que al presionar los botones y hacerlo de manera más lenta el botón captaba una pulsación repetida realizando más peticiones de las debidas al servidor. Este problema también se resolvió de manera eficiente.

7.2 Validación

El objetivo de la validación es la comprobación de que la funcionalidad requerida, es decir, los requisitos funcionalidades especificados con anterioridad, se cumplen y funcionan correctamente.

7.2.1 Desarrollo de validación

Con el fin de comprobar si la totalidad de los requisitos tanto funcionales como no funcionales eran cumplidos por el sistema, se realizó un estudio con usuarios con un perfiles similares a los de los usuarios finales.

Los usuario se repartieron en dos grupos, usuarios con un perfil similar al del usuario final de tipo dependiente y usuarios con un perfil similar al del usuario final de tipo responsable. Siendo el primer grupo formado por 2 usuarios de 83 y 84 años y el segundo formado por 4 usuarios de 36, 20, 32 y 40 años.

Se observó a los usuarios interactuar con el sistema durante aproximadamente 15 minutos cada uno y se evaluó mediante un listado que requisitos se habían cumplido y que requisitos no.

Los requisitos funcionales fueron cumplidos en su totalidad (10 de 10).Mientras que los requisitos no funcionales solo fueron cumplidos 7 de 8.

Tras una la investigación y solución de la incidencia, y nuevo paso de la prueba con usuarios, todos los requisitos fueron marcados como cumplidos, finalizando así el proceso de validación.

8 Evaluación

8.1 Evaluación de los usuarios

Los diferentes usuarios, tanto del grupo de usuarios dependientes como del grupo de usuarios responsables, han constatado la utilidad de la aplicación y las mejoras respecto a otros sistemas.

Una vez expuestas los objetivos del desarrollo de este proyecto y las metas de superación respecto a otros sistemas, los usuarios han confirmado la superación y cobertura de los objetivos iniciales.

Los usuarios han denotado mejora en la usabilidad respecto a otros sistemas del mercado, exponiendo la claridad de exposición de la información relevante.

La funcionalidad clave, que desde el planteamiento inicial del proyecto era la interacción mediante la voz con los usuarios dependientes, ha sido aprobada por todos los usuarios de prueba, agradeciendo en todo momento la mínima interacción con la pantalla táctil.

La capacidad de aprendizaje del sistema ha sido constatada por los usuarios responsables de prueba, y apreciada fuertemente por los usuarios menos ajenos a la tecnología.

En todo momento se ha planteado por todos los usuarios de prueba la comunicación de la información obtenida de su uso a un médico, lo que será contemplado como mejora futura del sistema.

8.1.1 Beneficios

Los beneficios a nivel social del sistema Mooi son tangibles, ya que es la única aplicación gratuita que ofrece un sistema de tele-asistencia que, aunque con unos servicios reducidos, alarga el tiempo de independencia de los usuarios de avanzada de edad.

Representa una mejora frente a otros sistemas sobre todo en usabilidad, ya que tras el estudio realizado, se ha identificado que es el único sistema que incorpora la interacción por voz y el mantenimiento de un flujo de información hacia los familiares responsables del usuario dependiente.

Uno de los beneficios destacables es la orientación académica de la recolección de datos derivados del uso de la aplicación ya que la información obtenida de la interacción de la aplicación con los usuarios de manera verbal se almacena para futuras explotaciones, tanto en el estudio de los pacientes como en el estudio de la relación humana con la inteligencia artificial.

Además de los claros beneficios sociales, también hay que destacar el cúmulo de tecnologías diferentes usado en su desarrollo, el front-end de la aplicación rompe con el paradigma clásico de aplicación web estática y limitada a los navegadores web, ya que puede ser utilizado en cualquier dispositivo móviles de los mencionados con anterioridad.

9 Conclusiones y líneas futuras

9.1 Conclusiones

Es este trabajo de fin de grado se ha realizado un trabajo de ingeniería del software sobre el proyecto de una aplicación móvil híbrida.

Desde su concepción, pasando por el estudio del arte, definición de funcionalidades y requisitos, selección de arquitectura, hasta su desarrollo en el cual se han realizado iteraciones de manera controlada gracias a su modelo en cascada iterativo.

Tras el estudio del arte realizado se detectó la escasez de dispositivos de tele-asistencia en forma de aplicaciones, y una falta absoluta de este tipo de sistemas de forma gratuita, además se apreció la carencia de usabilidad en este tipo de sistemas, cuya importancia es crucial debido a la escasa relación del sector social afectado (personas de la tercera edad) con la tecnología. Por ello se decidió desarrollar el sistema Mooi que cubre las carencias detectadas en este sector.

Su desarrollo ha supuesto un avance en cuanto al concepto que se tenía hasta ahora de los sistemas de tele-asistencia, dando a conocer las posibilidades de la integración de este tipo de sistemas con las tecnologías más novedosas.

Además no solo la naturaleza de las tecnologías utilizadas, si no también su cantidad, es un avance en cuanto a integración en aplicaciones híbridas.

El desarrollo de las labores de ingeniería del software han proporcionado a la estudiante conocimientos sobre la gestión de proyectos desde su concepción hasta su fin, y el desarrollo de la aplicación en sí ha dotado a la estudiante de numerosos conocimientos sobre tecnologías vanguardistas, así como de conocimientos sobre el mismo proceso de aprendizaje, dotándole de recursos para su vida profesional.

El trabajo conjunto de gestión y desarrollo del sistema ha requerido una gran implicación de la estudiante, muy superior a las 300 horas asignadas al desarrollo de un trabajo de fin de grado.

9.2 Trabajo futuro

Durante el desarrollo de la aplicación se han detectado carencias de los sistemas actuales no detectadas durante el proceso de definición de la aplicación. Además, los usuarios que han realizado las pruebas han aportado ideas interesantes sobre funcionalidades que ellos, los primeros afectados, han visto necesarias en un sistema como Mooi.

En siguientes iteraciones se ha propuesto en el medio plazo la inclusión de las siguientes funcionalidades:

- Inclusión de un perfil de la aplicación específico para médicos, con la capacidad de notificar actualizaciones sobre los perfiles de usuario no solo al usuario responsable, si no también a un usuario médico responsable.

- Inclusión de métodos de inserción para usuarios discapacitados, como ciegos, sordos etc. Incluyendo un sistema de selección de sistemas de manera intuitiva un usuario sordomudo podría conseguir definir una alarma con cierto grado de detalle.

Como iteraciones a largo plazo las funcionales adicionales a implementar son las siguiente:

- Obtención de información médica de los usuarios dependientes a través de un suministro externo que ceda información del historial médico del paciente o, en caso de no ser posible, obtención de información más completa del usuario a través de un formulario más detallado para rellenar por parte de los usuarios.

- Iteración sobre la generación de respuestas por parte de la inteligencia artificial de Mooi, de manera que no solo entienda mejor las respuestas del usuario, si no que elabore de manera autónoma diferentes respuestas a las fijadas en el desarrollo.

- Creación de un chat para la interacción entre usuarios dependientes y usuarios responsables integrando en dicho chat la respuesta mediante voz siguiente las líneas de usabilidad definidas en la aplicación.

10 Referencias

[1] Vodafone, « Teleasistencia Vodafone », [En línea]. Available: https://www.vodafoneteayuda.es/2015/01/teleasistencia-movil-para-emergencias-de-cruz-roja/. [Último acceso: 2016].

[2] SAR QUAVITAE, «Teleasistencia móvil», [En línea]. Available: http://www.sarquavitae.es/teleasistencia/. [Último acceso: 2016].

[3] Comunidad de Madrid, «Telealarma», [En línea]. Available: https://sede.madrid.es/portal/site/tramites/. [Último acceso: 2016].

[4] Apple, «SIRI», [En línea]. Available: http://www.apple.com/es/ios/siri/. [Último acceso: 2016].

[5] Microsoft, «CORTANA», [En línea]. Available: https://www.microsoft.com/es-es/windows/Cortana/. [Último acceso: 2016].

[6] El Pais, «La pensión media en España roza los 900 euros en marzo», [En línea]. Available: http://economia.elpais.com/economia/2016/03/22/actualidad/1458644632_840364.html. [Último acceso: 2016].

[7] Procesosoftware, «Modelo iterativo», [En línea]. Available: https://procesosoftware.wikispaces.com/Modelo+Iterativo?responseToken=bb2312cf4c0de6311463c4e33402ee6c. [Último acceso: 2016].

[8] IntelliJ, «Android Studio», [En línea]. Available: https://developer.android.com/studio/index.html?hl=es. [Último acceso: 2016].

[9] Apple, «XCode», [En línea]. Available: https://itunes.apple.com/es/app/xcode/id497799835?mt=12. [Último acceso: 2016].

[10] Phonegap, «Cordova», [En línea]. Available: https://cordova.apache.org/. [Último acceso: 2016].

[11] Mongodb Inc., «Mongodb», [En línea]. Available: https://www.mongodb.com/company/. [Último acceso: 2016].

[12] Dmitry Schetnikovich, «Robomongo», [En línea]. Available: https://robomongo.org//. [Último acceso: 2016].

[13] Node Fundation, «Nodejs», [En línea]. Available: https://nodejs.org/en/ . [Último acceso: 2016].

[14] Google, «Angularjs», [En línea]. Available: https://angularjs.org/. [Último acceso: 2016].

[15] Drifty Co, «Ionic», [En línea]. Available: http://ionicframework.com/. [Último acceso: 2016].

[16] Poiuytrez, «SpeechRecognizer», [En línea]. Available: https://github.com/poiuytrez/SpeechRecognizer. [Último acceso: 2016].

[17] Domaemon, «Text To Speech», [En línea]. Available: https://github.com/domaemon/org.apache.cordova.plugin.tts. [Último acceso: 2016].

[18] Daffl, «Natural Brain», [En línea]. Available: https://github.com/daffl/natural-brain/. [Último acceso: 2016].

[19] Apidoc, «APIDOC», [En línea]. Available: https://github.com/apidoc/apidoc/. [Último acceso: 2016]